APPENDIX A

PINECONE.H

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gl/gl.h>
#include <gl/device.h>
```

FREQS.H

```
#define DC_VALUE_RATIO 0.2

long number_freqs = 4;

double freq[10] = {
        1.9,
        1.654,
        1.476,
        1.227
};

double phase[10] = {
        2.111,
        0.0765,
        1.32,
        1.0,
        2.38,
        0.73,
        3.0,
        1.1
};
```

STAMP_IT.C

```
/* this program explores the specifics of stamping an image with 'digital
reticles' for the purpose of
determining registration between a given image and the digimarc signatures,
i.e., the registration
requirements for finding the scale and rotation elements of the ubiquitous
snowy image patterns */


#include "pinecone.h"
#include "freqs.h"

#define DISPLAY_IT 1
#define PI 3.141592653589
#define SQRT2_2 0.707106781186
#define INITIAL_SCALE 10.0

unsigned char *img;
long xdim,ydim;
Colorindex *disp;
Colorindex *scale_disp;
Colorindex *pdisp;

int helplines = 2;
char    *help[] =
{
        "usage: stamp_it filename xdim ydim channels \n",
        "\n stamp_it modifies the input image to include digital reticles and
outputs filename.stamped "
};


int load_scale_disp(void){
long i,j;

        pdisp = scale_disp+20;
```

```
        for(j=0;j<15;j++){
                for(i=1;i<20;i++){
                        *pdisp = (unsigned char)255;
                        pdisp += 20;
                }
                pdisp += 20;
        }
        pdisp = scale_disp+20+(35*400);
        for(j=0;j<15;j++){
                for(i=1;i<20;i++){
                        *pdisp = (unsigned char)255;
                        pdisp += 20;
                }
                pdisp += 20;
        }
return(0);
}

int draw_scale(double value){
long i;
double dtemp;

value *=10.0;

        memcpy(disp,scale_disp,40000);
        dtemp = 20.0 * value;
        if(dtemp > 399.0)dtemp=399.0;
        pdisp = &disp[15*400 + (int)dtemp];
        for(i=0;i<20;i++){
                *pdisp = (unsigned char)255;
                pdisp += 400;
        }
        rectwrite(0,0,399,49,disp);

return(0);
}
```

```
double calculate_change( double distance , double scale ,  long which ){
        long i;
        double value=DC_VALUE_RATIO * scale;

        for(i=0;i<number_freqs;i++){
                value += scale * sin ( freq[i] * distance + phase[i] +
(double)which * PI );
        }
        if(value < 0.0)value = 0.0;

return(value);
}


main( argc, argv )
int     argc ;
char    *argv[] ;
{
        long i,j,go=1,button,channels,which,count;
        long temp,increment,last_middle;
        unsigned char tmp,*pimg,*pimg1,*img_out,*img_r,*img_b;
        char string[80], outfile[80];
        FILE *inf;
        double change,current_scale = INITIAL_SCALE;
        long gid_img,gid_stamped,gid_scale;

        if(argc!=5) {
                for(j=0;j<helplines;j++)fprintf( stderr, "%s", help[j]) ;
                exit( 1 ) ;
        }

        xdim = atoi(argv[2]);
        ydim = atoi(argv[3]);
        channels = atoi(argv[4]);
        if(channels != 1 && channels !=3){
                fprintf( stderr, "stamp_it : channels must equal 1 for B/W or
3 for color\n" ) ;
                exit( 1 ) ;
```

```
        }

        if(DISPLAY_IT) disp = calloc(xdim*ydim, sizeof(Colorindex) ) ;
        else disp = calloc(40000, sizeof(Colorindex) ) ;
5       scale_disp = calloc(40000, sizeof(Colorindex) ) ;
        img = calloc(xdim*ydim, sizeof(unsigned char) ) ;
        img_out = calloc(xdim*ydim, sizeof(unsigned char) ) ;
        if( !disp || !scale_disp || !img || !img_out ){
                fprintf( stderr, "stamp_it : can not allocate space\n" ) ;
10              exit( 1 ) ;
        }

        if(channels == 3){
                img_r = calloc(xdim*ydim, sizeof(unsigned char) ) ;
15              img_b = calloc(xdim*ydim, sizeof(unsigned char) ) ;
                if( !img_r || !img_b ){
                        fprintf( stderr, "stamp_it : can not allocate space\n"
) ;
                        exit( 1 ) ;
20              }
        }

        /* read in binary data into array */
25      inf = fopen(argv[1],"r");
        if(!inf) {
                fprintf(stderr,"stamp_it: can't open %s\n",argv[1]);
                exit(1);
        }
30      if(channels == 3){
                fread(img_r,sizeof(unsigned char),xdim*ydim,inf);
                fread(img,sizeof(unsigned char),xdim*ydim,inf);
                fread(img_b,sizeof(unsigned char),xdim*ydim,inf);
        }
35      else fread(img,sizeof(unsigned char),xdim*ydim,inf);
        fclose(inf);

        /* flip it */
        pimg = img;
```

```
        pimg1 = &img[xdim*(ydim-1)];
        for(i=0;i<(ydim/2);i++){
                for(j=0;j<xdim;j++){
                        tmp = *pimg;
                        *(pimg++) = *pimg1;
                        *(pimg1++) = tmp;
                }
                pimg1 -= (2*xdim);
        }


        if(DISPLAY_IT){
                foreground();
                prefsize(xdim,ydim);
                gid_img = winopen("Original");
                gflush();

                pdisp = disp;
                pimg = img;
                for(i=0;i<(ydim*xdim);i++){
                                *(pdisp++) = (Colorindex)*(pimg++);
                }
                rectwrite(0,0,xdim-1,ydim-1,disp);

                prefsize(xdim,ydim);
                gid_stamped = winopen("Stamped Image...");
                rectwrite(0,0,xdim-1,ydim-1,disp);

                load_scale_disp();
                prefsize(400,50);
                gid_scale = winopen("Rough scaling...");
        }



        /* main visual feedback loop */
        last_middle = 0;
        while(go){
```

```
                /* first diagonal */
                for(i=0;i<(xdim+ydim-1);i++){
                        which = 0;
                        /* calculate addition or subtraction to image */
                        change = calculate_change( (double)i * SQRT2_2 ,
current_scale,  which );
                        if( i < xdim ){
                                pimg = &img[i*xdim];
                                pimg1 = &img_out[i*xdim];
                                count = i+1;
                        }
                        else {
                                pimg = &img[xdim*ydim - ydim - xdim + i + 1];
                                pimg1 = &img_out[xdim*ydim - xdim - ydim + i +
1];
                                count = xdim + ydim - i - 1;
                        }
                        for(j=0;j<count;j++){
                                temp = (long)*pimg + (long)(change+0.5);
                                if(temp > 255)temp = 255;
                                *pimg1 = (unsigned char)temp;
                                pimg -= (xdim-1);
                                pimg1 -= (xdim-1);
                        }
                }
                /* second diagonal */
                for(i=0;i<(xdim+ydim-1);i++){
                        which = 1;
                        /* calculate addition or subtraction to image */
                        change = calculate_change( (double)(xdim - 1 - i) *
SQRT2_2 , current_scale, which );
                        if( i < xdim ){
                                pimg1 = &img_out[xdim - i - 1];
                                count = i+1;
                        }
                        else {
                                pimg1 = &img_out[(i-xdim+1)*xdim];
                                count = xdim+ydim-i-1;
                        }
```

```
                for(j=0;j<count;j++){
                        temp = (long)*pimg1 + (long)(change+0.5);
                        if(temp > 255)temp = 255;
                        *pimg1 = (unsigned char)temp;
                        pimg1 += (xdim+1);
                }
        }


        if(DISPLAY_IT){
                pdisp = disp;
                pimg = img_out;
                for(i=0;i<(ydim*xdim);i++){
                        *(pdisp++) = (Colorindex)*(pimg++);
                }
                winset(gid_stamped);
                rectwrite(0,0,xdim-1,ydim-1,disp);

                pdisp = disp;
                pimg = img;
                for(i=0;i<(ydim*xdim);i++){
                        *(pdisp++) = (Colorindex)*(pimg++);
                }
                winset(gid_img);
                rectwrite(0,0,xdim-1,ydim-1,disp);

                winset(gid_scale);
                draw_scale(current_scale);
        }

        button=0;
        while(!button){
                if( getbutton(LEFTMOUSE) ){
                        current_scale *= 1.15;
                        button = 1;
                        last_middle = 0;
                }
                if( getbutton(RIGHTMOUSE) ){
                        current_scale *= 0.85;
```

```
                        button = 1;
                        last_middle = 0;
                }
                if( getbutton(MIDDLEMOUSE) ){
                        while( getbutton(MIDDLEMOUSE) );
                        if( last_middle ){
                                button = 1;
                                go=0;
                        }
                        last_middle = 1;
                }
        }
}

/* now re-sign output image because of slight changes to master key */

/* flip output image */
pimg = img_out;
pimg1 = &img_out[xdim*(ydim-1)];
for(i=0;i<(ydim/2);i++){
        for(j=0;j<xdim;j++){
                tmp = *pimg;
                *(pimg++) = *pimg1;
                *(pimg1++) = tmp;
        }
        pimg1 -= (2*xdim);
}

/* write out signed image */
sprintf(outfile,"%s.stamped",argv[1]);
inf = fopen(outfile,"w");
if(!inf) {
        fprintf(stderr,"stamp_it: can't open %s\n",outfile);
        exit(1);
}
if(channels == 3){
        fwrite(img_r,sizeof(unsigned char),xdim*ydim,inf);
        fwrite(img_out,sizeof(unsigned char),xdim*ydim,inf);
```

```
                fwrite(img_b,sizeof(unsigned char),xdim*ydim,inf);
        }
        else fwrite(img_out,sizeof(unsigned char),xdim*ydim,inf);
        fclose(inf);

        /* free and clean up */
        if(DISPLAY_IT) gexit();
        free(img);
        free(img_out);
        free(disp);
        free(scale_disp);
        if( channels = 3){
                free(img_r);
                free(img_b);
        }

}
```

REGISTR.C

```
/* this program is a companion to stamp_it, wherein it is given a suspect
image and it attempts to determine where
the cross-hatch pattern resides within the suspect image, thereby determining
the scale, roation and offset
of the suspect image */


#include "pinecone.h"
#include "freqs.h"

#define DISPLAY_IT 1
#define DISP_POW 0.1
#define MOV_AV   21     /* keep this odd please */
#define MAGG_THRESHOLD 1.7
#define ANGLE_INCREMENT (PI/360.0)
#define START_SCALE_ZONE 50
#define SCALE_START 0.5
#define SCALE_STOP 2.0
#define SCALE_STEP 0.001
#define MAX_BLOCK_SIZE 5
#define PI 3.141592653589

unsigned char *img;
long xdim,ydim;
Colorindex *disp;
Colorindex *pdisp;
long bits,fftdim,fft_size;
float *ar,*ai;
float wr[10000],wi[10000];

int helplines = 2;
char    *help[] =
{
        "usage: register filename xdim ydim channels \n",
        "\n register looks at the input image for digital reticles and
displays registered result "
};
```

```
int shift_array(float *array,int dim){
        int i,j;
        int dim2 = dim/2;
        int offset = dim2*dim + dim2;
        float *p1,*p2,ftmp;

        for(i=0;i<dim2;i++){
                p1 = &array[i*dim];
                p2 = &array[offset+i*dim];
                for(j=0;j<dim2;j++){
                        ftmp = *p1;
                        *p1 = *p2;
                        *p2 = ftmp;
                        p1++;p2++;
                }
        }
        offset = dim2*dim;
        for(i=0;i<dim2;i++){
                p1 = &array[dim2+i*dim];
                p2 = &array[offset+i*dim];
                for(j=0;j<dim2;j++){
                        ftmp = *p1;
                        *p1 = *p2;
                        *p2 = ftmp;
                        p1++;p2++;
                }
        }
        return(0);
}

int block_filter(float *array,int dim){
        int i,j,k,l,i2;
        float buffer[2][4096];

        for(i=0;i<dim;i++)buffer[0][i] = array[i];
        for(i=1;i<(dim-1);i++){
                i2 = i%2;
                buffer[i2][0] = array[i*dim];
                buffer[i2][dim-1] = array[i*dim+dim-1];
```

```
                for(j=1;j<(dim-1);j++){
                        buffer[i2][j]=0.0;
                        for(k=-1;k<2;k++){
                                for(l=-1;l<2;l++){
                                        buffer[i2][j]+= array[(i+k)*dim +
(j+1)];
                                }
                        }
                        buffer[i2][j]/=9.0;
                }
                i2 = (i-1)%2;
                for(j=0;j<dim;j++)array[(i-1)*dim + j] = buffer[i2][j];
        }
        i2 = (i-1)%2;
        for(j=0;j<dim;j++)array[(i-1)*dim + j] = buffer[i2][j];

        return(0);
}


/* assumes fftdim arrays ar and ai */
double get_mag(double x, double y){
        long xoff,yoff;
        float *par,*pai;
        double xdist,ydist,cf,r_value,i_value,value=0.0;

        xoff = (long)x;
        yoff = (long)y;
        xdist = x - (double)xoff;
        ydist = y - (double)yoff;

        par = &ar[yoff * fftdim + xoff];
        pai = &ai[yoff * fftdim + xoff];

        cf = (1.0 - xdist) * (1.0-ydist);
        r_value = (double)*(par++);
        i_value = (double)*(pai++);
```

```
        value = cf * sqrt(r_value*r_value + i_value*i_value);
        cf = xdist * (1.0-ydist);
        r_value =  (double)*par;
        i_value =  (double)*pai;
        value += cf * sqrt(r_value*r_value + i_value*i_value);
        par += (fftdim-1);
        pai += (fftdim-1);

        cf = (1.0 - xdist) * ydist;
        r_value = (double)*(par++);
        i_value = (double)*(pai++);
        value += cf * sqrt(r_value*r_value + i_value*i_value);
        cf = xdist * ydist;
        r_value = (double)*par;
        i_value = (double)*pai;
        value += cf * sqrt(r_value*r_value + i_value*i_value);

        return(value);
}



main( argc, argv )
int     argc ;
char    *argv[] ;
{
        long i,j,k,go,channels,count,center;
        long dim,angle_int,total_angles,top_candidate;
        unsigned char tmp,*pimg,*pimg1,*img_out,*img_r,*img_b;
        char string[80], outfile[80];
        FILE *inf;
        long gid_img;
        double
x,y,coss,sinn,angle,mag,magg[5000],magg_ma[5000],angle_vector[10000];
        double radius,dtmp,dscale,grey_diff,highest,frac,highest_scale,scale;
        float *par,*pai;
```

```
        if(argc!=5) {
                for(j=0;j<helplines;j++)fprintf( stderr, "%s", help[j]) ;
                exit( 1 ) ;
        }

        xdim = atoi(argv[2]);
        ydim = atoi(argv[3]);
        channels = atoi(argv[4]);
        if(channels != 1 && channels !=3){
                fprintf( stderr, "register : channels must equal 1 for B/W or
3 for color\n" ) ;
                exit( 1 ) ;
        }

        /* find the next power of two equal to or higher than the highest
input dimension */
        if(xdim > ydim)dim = xdim;
        else dim = ydim;
        fftdim = 1; go = 1; bits = 0;
        while( go ){
                if( dim > fftdim ){
                        fftdim*=2;
                        bits++;
                }
                else go = 0;
        }
        if(bits > 12){
                fprintf( stderr, "recognize : sorry, this particular program
only accepts 4K images and less\n" ) ;
                exit( 1 ) ;
        }
        fft_size = fftdim * fftdim;

        disp = calloc(fft_size, sizeof(Colorindex) ) ;
        img = calloc(xdim*ydim, sizeof(unsigned char) ) ;
        ar = calloc(fft_size, sizeof(float) ) ;
        ai = calloc(fft_size, sizeof(float) ) ;
        if( !disp || !img || !ar || !ai ){
                fprintf( stderr, "register : can not allocate space\n" ) ;
```

```
                exit( 1 ) ;
        }

        if(channels == 3){
                img_r = calloc(xdim*ydim, sizeof(unsigned char) ) ;
                img_b = calloc(xdim*ydim, sizeof(unsigned char) ) ;
                if( !img_r || !img_b ){
                        fprintf( stderr, "register : can not allocate space\n"
) ;
                        exit( 1 ) ;
                }
        }

        /* read in binary data into array */
        inf = fopen(argv[1],"r");
        if(!inf) {
                fprintf(stderr,"register: can't open %s\n",argv[1]);
                exit(1);
        }
        if(channels == 3){
                fread(img_r,sizeof(unsigned char),xdim*ydim,inf);
                fread(img,sizeof(unsigned char),xdim*ydim,inf);
                fread(img_b,sizeof(unsigned char),xdim*ydim,inf);
        }
        else fread(img,sizeof(unsigned char),xdim*ydim,inf);
        fclose(inf);

        /* flip it */
        pimg = img;
        pimg1 = &img[xdim*(ydim-1)];
        for(i=0;i<(ydim/2);i++){
                for(j=0;j<xdim;j++){
                        tmp = *pimg;
                        *(pimg++) = *pimg1;
                        *(pimg1++) = tmp;
                }
                pimg1 -= (2*xdim);
        }
```

```
        /* copy image buffer into ar */
        par = ar;
        pimg = img;
        for(i=0;i<ydim;i++){
                for(j=0;j<xdim;j++){
                        *(par++) = (float)*(pimg++);
                }
                par += (fftdim-xdim);
        }

        /* 2d fft, in place */
        printf("\nforward fft...  \n");
        fft2d(ar,ai,bits,0,wr,wi);
        printf("done \n.... ");
        shift_array(ar,fftdim);
        shift_array(ai,fftdim);
        center = fftdim/2;
/*
        block_filter(ar,fftdim);
        block_filter(ai,fftdim);
*/

        if(DISPLAY_IT){
                foreground();
                prefsize(fftdim,fftdim);
                gid_img = winopen("yahh");
                gflush();

                dtmp =
(double)ar[center*fftdim+center+1]*(double)ar[center*fftdim+center+1];
                dtmp +=
(double)ai[center*fftdim+center+1]*(double)ai[center*fftdim+center+1];
                dscale = pow(dtmp, DISP_POW);

                pdisp = disp;
                par = ar;
                pai = ai;
                for(i=0;i<(fftdim*fftdim);i++){
```

```
                                dtmp = pow( (*par * *par + *pai * *pai) ,
DISP_POW );
                                dtmp *= (255.0 / dscale);
                                if(dtmp>255.0)dtmp = 255.0;
                                *(pdisp++) = (Colorindex)( dtmp  );
                                par++;pai++;
                        }
                        rectwrite(0,0,fftdim-1,fftdim-1,disp);
                }

                /* now search for the gross rotation axes */
                for(angle = 0.0,angle_int = 0;angle<PI/2;angle+=ANGLE_INCREMENT,
angle_int++){
                        coss = cos(angle);
                        sinn = sin(angle);
                        /* fill radial vector */
                        for(i=0;i<(fftdim/2);i++){
                                radius = (double)i;
                                x = (double)center - radius * coss;
                                y = (double)center - radius * sinn;
                                mag = get_mag(x,y);
                                x = (double)center + radius * sinn;
                                y = (double)center - radius * coss;
                                mag += get_mag(x,y);
                                magg[i] = mag;
                        }
                        /* create moving average */
                        magg_ma[MOV_AV/2] = 0.0;
                        for(i=0;i<MOV_AV;i++){
                                magg_ma[MOV_AV/2] += magg[i];
                        }
                        magg_ma[MOV_AV/2] /= ( (double)MOV_AV );
                        for(i=(MOV_AV/2)+1;i<(fftdim/2)-(MOV_AV/2)-1;i++){
                                magg_ma[i] = magg_ma[i-1];
                                magg_ma[i] -= ((magg[i - (MOV_AV/2) -
1])/(double)MOV_AV);
                                magg_ma[i] += ((magg[(MOV_AV/2) + i])/(double)MOV_AV);
                        }
```

```
                /* within prescribed 'scale zone', calculate final number for
this angle */
                angle_vector[angle_int] = 0.0;
                for(i=START_SCALE_ZONE;i<(fftdim/2) - (MOV_AV/2) -1;i++){
                        mag = magg[i] / magg_ma[i];
                        if( mag > MAGG_THRESHOLD ){
                                mag -= MAGG_THRESHOLD;
                                angle_vector[angle_int] += (mag*mag);
                        }
                }
        }
        total_angles = angle_int;

        /* sort out the TOP_CANDIDATES and find which has the best match on
absolute scale */
        /* chhose the highest angle_vector number for starters */
        highest = 0.0;
        for(angle_int=0;angle_int<total_angles;angle_int++){
                if(angle_vector[angle_int]>highest){
                        highest = angle_vector[angle_int];
                        top_candidate = angle_int;
                }
        }

        printf("\n\n tilt from original found = %d  ", (top_candidate/2) -
45);

        coss = cos(ANGLE_INCREMENT * (double)top_candidate);
        sinn = sin(ANGLE_INCREMENT * (double)top_candidate);
        /* fill radial vector for this angle */
        for(i=0;i<(fftdim/2);i++){
                radius = (double)i;
                x = (double)center - radius * coss;
                y = (double)center - radius * sinn;
                mag = get_mag(x,y);
                x = (double)center + radius * sinn;
                y = (double)center - radius * coss;
                mag += get_mag(x,y);
                magg[i] = mag;
```

```
        }
        /* create moving average */
        magg_ma[MOV_AV/2] = 0.0;
        for(i=0;i<MOV_AV;i++){
                magg_ma[MOV_AV/2] += magg[i];
        }
        magg_ma[MOV_AV/2] /= ( (double)MOV_AV );
        for(i=(MOV_AV/2)+1;i<(fftdim/2)-(MOV_AV/2)-1;i++){
                magg_ma[i] = magg_ma[i-1];
                magg_ma[i] -= ((magg[i - (MOV_AV/2) - 1])/(double)MOV_AV);
                magg_ma[i] += ((magg[(MOV_AV/2) + i])/(double)MOV_AV);
        }
        /* now slide the scale and find the highest point */
        highest = 0.0;
        for(scale = SCALE_START;scale < SCALE_STOP;scale+=SCALE_STEP){
                mag = 0.0;
                for(j=0;j<number_freqs;j++){
                        radius = scale * freq[j] * (double)fftdim / PI / 2.0;
                        if( (int)radius <= (1+MOV_AV/2) || (int)(radius+1) >=
((fftdim/2) - (MOV_AV/2) - 1) );
                        else {
                                frac = radius - (double)( (int) radius);
                                mag += (1.0-frac)*(magg[ (int)radius ] /
magg_ma[ (int)radius ]);
                                mag += frac*(magg[ (int)(radius+1) ] /
magg_ma[ (int)(radius+1) ]);
                        }
                }
                if(mag > highest){
                        highest = mag;
                        highest_scale = scale;
                }
        }
        printf("\n\nscale found = %lf   \n",1.0/highest_scale);

        /* now find the exact offset and orietnation, i.e. if it is flipped,
etc. */
```

```
        /* display the result at correct rotation and pixel size */

    sleep(1000);


        /* free and clean up */
        if(DISPLAY_IT) gexit();
        free(img);
        free(disp);
        free(ar);
        free(ai);
        if( channels = 3){
                free(img_r);
                free(img_b);
        }

    }
```